A Comprehensive Framework for Synthesizing Stencil Algorithms on FPGAs using OpenCL Model

Shuo Wang, Yun Liang*
Center for Energy-Efficient Computing and Applications (CECA), School of EECS, Peking University, China
{shvowang, ericlyun}@pku.edu.cn

ABSTRACT

Iterative stencil algorithms find applications in a wide range of domains. FPGAs have long been adopted for computation acceleration due to its advantages of dedicated hardware design. Hence, FPGAs are a compelling alternative for executing iterative stencil algorithms. However, efficient implementation of iterative stencil algorithms on FPGAs is very challenging due to the data dependencies between iterations and elements in the stencil algorithms, programming hurdle of FPGAs, and large design space.

In this paper, we present a comprehensive framework that synthesizes iterative stencil algorithms on FPGAs efficiently. We leverage the OpenCL-to-FPGA toolchain to generate accelerator automatically and perform design space exploration at the high level. We propose to bridge the neighboring tiles through pipe and enable data sharing among them to improve computation efficiency. Then, we extend the equal tile size design to a heterogeneous design with different tile size to balance the computation among different tiles. We also develop analytical performance models to explore the complex design space. Experiments using a wide range of stencil applications demonstrate that on average our heterogeneous implementations achieve 1.65X performance speedup but with less hardware resource compared to the state-of-the-art.

1. INTRODUCTION

Iterative stencil applications are widely employed in a variety of different fields of application, ranging from high-performance scientific computing, image processing, and medical computing [1, 2, 3]. In general, the stencil algorithms are iteratively invoked until the desired number of iterations has been performed. The stencil algorithms are often structured in a way that each processing step operates on all the array elements and each array element is updated following a stencil update function using the neighboring elements.

As technology scaling is nearing the end, the specialized hardware accelerator is a promising solution to cope with the continuous demand for high performance and energy efficiency of stencil algorithms. FPGAs provides hardware performance with general programmability. Designers can create dedicated pipelines with parallel processing elements, customized bit width, etc. on FGPAs. More importantly, the FPGA design cost can be reduced by raising the programming abstraction from tedious RTL programming to high-level programming such as C, C++ [4, 5, 6, 7]. Therefore, there is a rapid increase in popularity of using FPGAs as accelerators. For example, Microsoft deployed FPGA-accelerated nodes to accelerate the Bing search service [8]; Baidu uses FPGAs to speed up the deep learning models [9]; JP Morgan uses FPGAs to accelerate risk analysis.

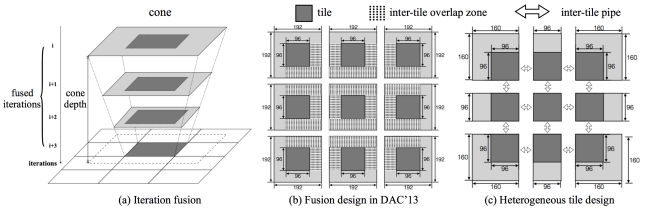


Figure 1: Comparison of Different Designs.

While of benefits of FPGA is clear, it is very challenging to design an efficient implementation of iterative stencil algorithms on FPGAs. In general, efficient implementation requires a thorough understanding of both application algorithm and hardware platform. On the application side, the stencil algorithms contain strict data dependencies across iterations and elements. On the hardware side, there exist different implementation choices exhibiting performance and hardware area trade-offs.

Iterative stencil algorithms inherently incur large memory transfer overhead due to the global memory synchronization after each iteration. To alleviate the global memory synchronization overhead, Nacci et al. [10] recently propose to fuse multiple iterations together for a tile and compute multiple tiles in parallel. As shown in Figure 1(a), for a tile computation, multiple iterations are fused to a *cone*. This design helps to reduce the global memory transfer as the intermediate data required by the *cone* can be accommodated on-chip. To ensure the data dependency, we have to surround the tile with extra elements required in the stencil pattern as shown in Figure 1(a). Thus, the actual area of computation for each tile is greater than the tile size as shown in Figure 1(b). Since each tile has its own *cone*, multiple tiles can be processed in parallel.

The iteration fusion based design is appealing. However, the overlapped region among neighboring tiles introduce redundant computations as shown in Figure 1(b) and the amount of the redundant computations increases with the depth of the *cone* and dimension of the stencils (e.g. 3D stencil). Therefore, in reality, the redundant computation may offset the reduction of global memory transfer and synchronization. In this paper, we propose to bridge the neighboring tiles through pipes and enable data sharing among them to reduce the redundant computation as illustrated in Figure 1(c). Each tile performs its own computation and uses pipes to share the common data with its neighboring tiles. This greatly reduces the redundant computation and saves

*Corresponding author

DAC'17, June 18-22, 2017, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/3061639.3062185

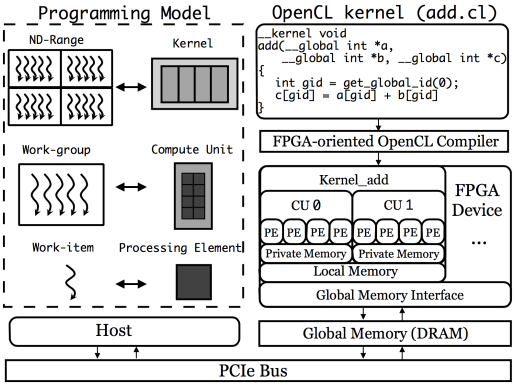


Figure 2: OpenCL programming model mapping to FPGA.

the on-chip resources. To ensure the data dependency, all the tiles must synchronize at the iteration barrier and thus the execution time is determined by the slowest tile (e.g., the tile at the corner) as it has the largest amount of computation. To solve this problem, we propose to use different tile size to balance their computation.

We develop an automatic tool flow based on Xilinx OpenCL-to-FPGA flow for iterative stencil applications. With an original stencil algorithm written in OpenCL as input, our framework can transform the OpenCL code to enable the iteration fusion, pipe-based data sharing, and heterogeneous tiling automatically. Our final heterogeneous designs exhibit a large design space to explore (e.g. *cone* depth, tile structure, etc). Hence, we develop analytical performance models to compare different designs and explore the design space. By doing this, we can identify the designs with the highest performance.

This work makes the following contributions,

- We propose a new heterogeneous architecture design for stencil algorithms to improve performance and save and FPGA resources.
- We develop an accurate performance model to help determine the optimal parameters of our proposed stencil accelerator design.
- We propose a framework that automatically optimizes and synthesizes iterative stencil algorithms onto FPGAs and validate our framework using a suite of stencil algorithms with different dimensions.

Experiments show that compared to the state-of-the-art, our heterogeneous design achieves on average 1.65X performance speedup but with less on-chip resources.

2. BACKGROUND AND RELATED WORK

In this section, we introduce OpenCL programming model to FPGA mapping, iterative stencil algorithm, and related work.

2.1 OpenCL to FPGA Mapping

OpenCL is a cross-platform programming model which employs a single-instruction multiple data (SIMD) model enabling implementation of general purpose programs on heterogeneous systems. From the perspective of a programmer, OpenCL has three levels of hierarchy, which are ND-Range kernel, work-group, and work-item as shown in Figure 2. As for the execution, OpenCL host program dynamically invokes the ND-Range kernel to be executed a specific hardware kernel on FPGA. Multiple work-groups bundled together to form an ND-Range kernel are distributed to different compute units (CUs). The basic unit of execution is single work-item, which is executed on processing element (PE) in a pipeline fashion. A group of work-items is bundled together to form a work-group. OpenCL uses a relaxed memory consistency model for global memory within kernel's workspace. Work-items in a work-group can be synchronized together through an explicit barrier operation. Within a kernel, the execution order of different work-groups does not affect the output. The local memory is shared among the CUs within a kernel and cannot be directly accessed by the host.

```
for k = 0: num_iterations
    load Global_A to A;
    load Global_B to B;
    for all i = 0 : ROWS-1 and j = 0: COLS-1
        top = max(i-1,0);
        bottom = min(i+1, ROWS-1);
        left = max(j-1, 0);
        right = min(j+1, COLS-1);
        A[i][j] = B[i][j] + B[top][j] + B[bottom][j]
                  + B[i][left] + B[i][right];
    swap(A, B);
    write A to Global;
```
Figure 3: Jacobi-2D pseudo code.

2.2 Iterative Stencil Algorithm

Iterative stencil algorithm is a class of algorithms that iteratively update the values of array elements according to some fixed pattern, called stencil. Algorithms following this pattern are widely used in scientific simulations [1], image processing [2] and scientific computations [3]. As we can see the *Jacobi-2D* algorithm from Figure 3, the current element $A[i][j]$ is updated by its neighboring data of the previous iteration, and then the data are swapped out after each iteration to do synchronization.

2.3 Related Work

Performance optimization for iterative stencil algorithms has been studied for CPUs or GPUs [11, 12, 13, 14]. In these platforms, tiling is a widely employed technique to improve data locality in cache or workload balancing on multiple cores [11, 14]. In order to reduce the off-chip global memory access overhead, fused iteration is proposed for iterative stencil algorithms to increase the the on-chip computation to off-chip memory access ratio [15, 12].

In the context of reconfigurable devices such as FPGAs, most of the studies on stencil benchmarks are based on handwriting RTL code [16, 17]. [16] proposes a sliding-window based method to reduce BRAM consumption for stencil algorithms. [17] maps stencil algorithms to systolic array. Both methods, however, cannot alleviate the large off-chip global memory access overhead. Cong et al. [18] studies on fully pipelining the global memory access and stencil computations which require a large amount of reused buffers. To reduce the buffer size, a stencil-specific microarchitecture template is proposed.

Recently, the iteration fusion based optimization method has been used to reduce the off-chip memory access overhead for stencil [10] and deep neural networks [19]. Nacci et al. [10] first applied fused iteration fusion optimiztion for stencil algorithm on FPGA. The work proposes an architecture template-based optimization framework to automatically generate the optimized C code which is then implemented on FPGA by using commercial HLS tools. The experimental result is promising which achieves an order of

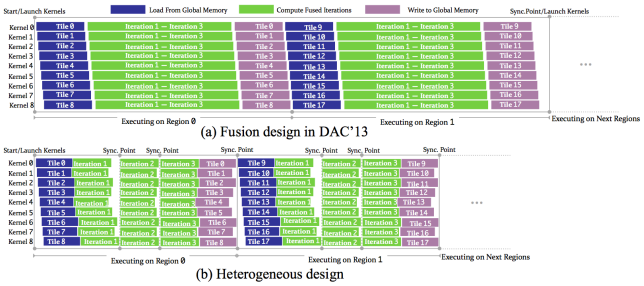


Figure 4: Kernel Execution of Different Designs.

magnitude higher performance for iterative gaussian filter and Chambolle algorithm [20].

3. HETEROGENEOUS DESIGN

In this section, we propose a heterogeneous architecture design which employs pipe-based data sharing and workload balancing techniques to solve the redundant computation and unbalanced workload problems.

3.1 Pipe-based Data Sharing

Redundant Computation Elimination. In the baseline design shown in Figure 4(a), each kernel could be executed independently within fused iterations by processing the redundant data of the neighboring tiles. In our heterogeneous design, we propose to use pipe to share the data near the boundary between adjacent kernels. Pipe is a new feature introduced in OpenCL 2.0 for passing data between kernels. As we can see from Figure 1(c), pipes are added for each adjacent pair of kernels. After the redundant computation is eliminated, the on-chip memory could also be saved because there is no need to reserve the storage resources for the redundant data. In the OpenCL-to-FPGA mapping, pipe is implemented as FIFOs which consumes much fewer on-chip memory resources compared with the redundant data.

Communication Latency Hiding. Although the redundant computation and on-chip memory have been eliminated, the data communication between adjacent computation kernels incurs latency overhead. To solve this problem, we propose a scheduling based strategy to hide the communication latency. The main idea is to do the interior computation first while simultaneously waiting for the boundary data updates. To be more specific, we categorize the elements in each tile into two groups, independent group and dependent group depending on whether it requires the data transferred from adjacent tiles to do the computation. In the working flow of the computation kernel, the independent elements are always given higher priorities. Since the pipe operations are executed in parallel with the processing of independent elements, the data sharing overhead can be partially or completely hidden.

3.2 Workload Balancing

As shown in Figure 4(b), for each fused iteration, the computation kernels compute the tiles simultaneously and must synchronize (write the data to global memory) before they move to the next region. Thus, the execution time of each region is constrained by the slowest kernel. In the equal tiling scheme, the computation workload could be severely unbalanced, because the redundant computations between the adjacent computation kernels are completely eliminated, the ratio of computation workload of the kernels near the boundary to the workload of the inner kernels are highly skewed. To alleviate the workload unbalancing problem, we propose to employ a balanced tiling strategy to balance the amount of workload among different kernels. The main idea is to employ a heterogeneous tiling method which decreases the tile size of the kernel near the boundary while increases the sizes of the rest of tiles.

Table 1: Summary of Analytical Model Parameters.

Model Parameter	Definition	Obtained
L	Execution latency of entire stencil algorithm	predicted by model
N_{region}	Number of regions given an input size	source code analysis
$L^{tile}_{krnl_k}$	Execution latency of k^{th} kernel $krnl_k$ to execute a tile	predicted by model
H	Number of input stencil iterations	source code analysis
h	Number of fused iterations	determined by model
D	Number of input stencil dimensions	source code analysis
K	Number of kernels working in parallel	source code analysis
I^k_d	Workload balancing factor of k^{th} kernel in the d^{th} dimension	determined by model
W_d	Length of input stencil array along d^{th} dimension	source code analysis
w_d	Length of tile of along d^{th} dimension	source code analysis
Δw_d	Incremental length of tile along d^{th} dimension per fused iteration	source code analysis
$L^{mem}_{krnl_k}$	Latency of k^{th} kernel consumed by global memory access within a region	predicted by model
$L^{comp}_{krnl_k}$	Latency of k^{th} kernel consumed by computation within a region	predicted by model
$L^{launch}_{krnl_k}$	Latency of k^{th} kernel consumed by kernel launches within a region	predicted by model
$L^{read}_{krnl_k}$, $L^{write}_{krnl_k}$	Latency of k^{th} kernel consumed by read from/ write to global memory	predicted by model
$Size^{read}_{data}$, $Size^{write}_{data}$	Size of data of one work-group to be read from/write to global memory	source code analysis
BW	Peak bandwidth of global memory	off-line profiling
Δs	Bit size of transferred data	source code analysis
$L^{iter_i}_{krnl_k}$	latency of k^{th} kernel to complete the computation workload of i^{th} iteration	predicted by model
$C_{element}$	Number of clock cycles per element	source code analysis
II	Initiation interval of pipeline	HLS report
N_{unroll}	loop unrolling number in stencil benchmark	source code analysis
$L^{share_i}_{krnl_k}$	Latency of k^{th} kernel to transfer all the data through pipes in i^{th} iteration	predicted by model
C_{pipe}	number of clock cycles consumed to transfer one data element	off-line profiling

4. ANALYTICAL MODEL

In order to analyze the benefits and limitations of the proposed heterogeneous architecture, we build an analytical performance model. We define the execution latency in clock cycles of a stencil application as L. We predict L by modeling the pipe-based data sharing, synchronization, global memory transfer, and computation. Table 1 lists all the symbols used in the analytical model.

4.1 Inter-Kernel Synchronization

In general, an entire stencil input is divided into regions. Each region may contain multiple tiles as shown in Figure 4 depending on the resource constraints and these tiles are processed in parallel. In our heterogeneous designs, multiple tiles in a region are processed in parallel but need to synchronize in the end before executing the next iteration as shown in Figure 4 (b). The synchronization mechanism ensures that the all the tiles will share the same and updated value. Since each tile processes a different area in the region, each tile corresponds to an OpenCL kernel code.

The execution time of a region is determined by the slowest kernel. Thus, L is calculated as follows,

$$L = N_{region} \cdot \max_{k=1}^{K} L^{tile}_{krnl_k}, \tag{1}$$

where N_{region} is the number of regions given an input stencil size, K is the number of tiles contained by a region, and $L^{tile}_{krnl_k}$ is the latency of kernel $krnl_k$ to execute the k^{th} tile. N_{region} is calculated as follows,

$$N_{region} = \frac{H}{h} \cdot \frac{\prod_{d=1}^{D} W_d}{K \cdot \prod_{d=1}^{D} w_d}, \tag{2}$$

where H is the total number of iterations of a specific iterative stencil algorithm, h is the number of fused iterations (*cone* depth), W_d and w_d are the length of input stencil array and tile along the d^{th} dimension, respectively, and D is the maximum dimension number of stencil input array.

For simplicity, we define $L^{max}_{krnl} = \max_{k=1}^{K} L^{tile}_{krnl_k}$ which can be calculated by summing the latency in memory, computation, and kernel launch of the slowest kernel.

$$L^{max}_{krnl} = L^{mem}_{krnl} + L^{comp}_{krnl} + L^{launch}_{krnl}, \tag{3}$$

where L_{krnl}^{max}, L_{krnl}^{comp}, and L_{krnl}^{launch} are the latencies of the slowest kernel in global memory access, computation and kernel launch, respectively.

4.2 Global Memory Transfer

Figure 4 shows that the computations are separated by global memory transfers for both designs, and thus the total latency of global memory transfers could be calculated by adding up the time consumed in loading data from and writing data back to global memory. Thus, the latency of global memory transfer is thus calculated as

$$L_{krnl}^{mem} = L_{krnl}^{read} + L_{krnl}^{write}, \tag{4}$$

where L_{krnl}^{read} and L_{krnl}^{write} are clock cycles in reading from and writing back to global memory, respectively. The read and write are done in burst mode for high throughput. The burst mode is always coupled with barriers, where the data of one work-group is bundled together to copy from global memory to local memory and the memory transfer has to complete before the barriers. In the burst mode, the global memory accesses are coalesced and this potentially will lead to high bandwidth utilization. Moreover, when multiple kernels are working simultaneously, the global memory bandwidth are evenly shared among different kernels. Thus, we have

$$L_{krnl}^{read} = \frac{Size_{data}^{read}}{BW/K} = \frac{\Delta s \cdot \prod_{d=1}^{D} (w_d \cdot f_d^{max} + \Delta w_d \cdot h)}{BW/K} \tag{5}$$

$$L_{krnl}^{write} = \frac{Size_{data}^{write}}{BW/K} = \frac{\Delta s \cdot \prod_{d=1}^{D} w_d \cdot f_d^{max}}{BW/K} \tag{6}$$

where $Size_{data}^{read}$ and $Size_{data}^{write}$ are the size of data in one region to be read from/write to global memory. BW is the peak bandwidth between global memory and FPGA, Δs is the size of the input data type (e.g. 32-bit for float type), w_d is the length of the tile along the d^{th} dimension, f_d^{max} is the balancing factor of the slowest kernel along the d^{th} dimension, and Δw_d is the incremental length of the tile along the d^{th} dimension per fused iteration.

4.3 Computation

To predict the execution time of a kernel when executing a tile, we need to model the fused iterations and the synchronization between iterations. As shown by Figure 4, the computation period of a tile contains multiple iterations and the computation workload decreases when the iteration increases. Thus, the execution time of a kernel is calculated by accumulating the execution time of each iteration and the overhead of data sharing as follows,

$$L_{krnl}^{comp} = (1 + \lambda_{krnl}^{iter_i}) \cdot \sum_{i=1}^{h} L_{krnl}^{iter_i}, \tag{7}$$

where $L_{krnl}^{iter_i}$ is the latency of the slowest kernel to complete the computation workload of i^{th} iteration, and $\lambda_{krnl}^{iter_i}$ ($0 \leq \lambda_{krnl}^{iter_i} \leq 1$) represents the proportion of overhead incurred by data sharing between adjacent tiles. The details about $\lambda_{krnl}^{iter_i}$ is discussed in the following section. $L_{krnl}^{iter_i}$ could be calculated by multiplying the number of clock cycles per element of a tile and the number of elements in this iteration as follows,

$$L_{krnl}^{iter_i} = C_{element} \cdot \prod_{d=1}^{D} ((w_d \cdot f_d^{max} + \Delta w_d \cdot (h - i)), \tag{8}$$

where $C_{element}$ is the number of clock cycles per element. $C_{element}$ is determined by the number of processing elements N_{PE}, which can be controlled by the designers using loop unrolling #pragma, and the initiation interval of stencil computation pipeline II. Thus, $C_{element}$ is calculated as follows,

$$C_{element} = II/N_{PE}. \tag{9}$$

4.4 Inter-tile Data Sharing

We employ pipe in OpenCL to exchange the common data shared by adjacent tiles as shown in Figure 1 (c) and (d). Pipe is a new feature introduced in OpenCL 2.0 for passing data between kernels. In the OpenCL-to-FPGA mapping, pipe is implemented as FIFOs. Different tiles transfer different amount of data. We first define the following metric,

$$L_{krnl}^{share_i} = C_{pipe} \cdot \sum_{j=1}^{D} \prod_{d=1, d \neq j}^{D} (w_d \cdot f_d^{max} - \Delta w_d \cdot (h - i)), \tag{10}$$

where $L_{krnl}^{share_i}$ is the latency of the slowest kernel to transfer all the data through pipes in i^{th} iteration and C_{pipe} is the number of clock cycles consumed to transfer one data element.

Compared with the computation time, the message passing latency is relatively small. In practice, we can partially or completely hide the message passing by overlapping it with computation (see Section 3.1). Therefore, we introduce λ to represent the overlapping ratio of the message passing, which is 0 if all the message passing operations are hidden by the computation and is 1 if none of the message passing operations are overlapped. Then, we compute $\lambda_{krnl}^{iter_i}$ as follows,

$$\lambda_{krnl}^{iter_i} = \begin{cases} 0, & L_{krnl}^{share_i} \leq L_{krnl}^{iter_i} \\ \frac{L_{krnl}^{share_i} - L_{krnl}^{iter_i}}{L_{krnl}^{iter_i}}, & L_{krnl}^{share_i} \geq L_{krnl}^{iter_i} \end{cases}. \tag{11}$$

5. OPTIMIZATION FRAMEWORK

In this section, we propose an automatic optimization framework shown in Figure 5 by first introducing how our performance optimizer finds the optimal parameters of our proposed heterogeneous designs and then elaborating the details of the automatic code generation.

5.1 Performance Optimizer

As shown in Figure 5, the performance optimizer is mainly composed of feature extractor and analytical model. The feature extractor inside performance optimizer is in charge of analyzing original stencil operation code and determining the application-specific stencil configurations (stencil shape, dimension, operation type). Global memory bandwidth BW and the number of parallel stencil kernel K are user-defined parameters and fed to our performance optimizer as inputs. The initiation interval II of the computation pipeline is obtained by using the FlexCL framework [21].

After extracting required features, all the required parameters are sent to the analytical model proposed in Section 4. To obtain the optimal configuration parameters for our heterogeneous design, we enumerate both the number of fused iterations h and load balancing factor f_d^k for heterogeneous designs to achieve the best performance.

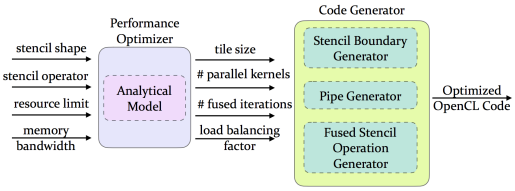


Figure 5: Overview of automatic optimization framework.

5.2 Automatic Code Generator

The automatic code generator has to adapt to application-specific configurations including dimension of input data, shape of stencil, type of stencil operation and etc.. For the OpenCL code of our proposed heterogeneous design, we split the code into three major parts, (1) stencil boundary, (2) data sharing pipes, (3) and fused stencil operation. Based on this, we propose an automatic code generator which could produce these three parts of code respectively and then merge them together into an OpenCL kernel.

Stencil Boundary Generator. For a specific stencil computation kernel, the stencil tile boundary varies at different iterations and is dependent on three factors, stencil shape, current iteration number and tile size. The stencil shape is obtained by our feature extractor, the tile size is user-defined value, and the iteration number is set to be a variable. So, for any stencil benchmark, the proposed stencil boundary generator could automatically generate a boundary as a function of stencil shape, tile size, and current iteration number.

Data Sharing Pipe Generator. The pipes are used to transfer the boundary data between adjacent kernels. But the pipe in OpenCL is one-directional, and thus we need to generate two pipes, read and write pipes, for each boundary of adjacent kernels. The stencil boundary generated by stencil boundary generator is also used as a reference for pipe generator to determine whether a data element should be transferred or not.

Fused Stencil Operation Generator. The original unoptimized stencil operation code and stencil boundary are used as input of fused stencil operation generator. First, the iteration fusion loop is added outside of the original stencil operation loop, and the loop boundary is provided by the stencil boundary generator. The data array residing off-chip global memory is then enhanced to local memory (BRAM) using the OpenCL *__local* declaration, and the local data array size is calculated by subtracting the shared data size from the data size according to the data boundary in the first level of fused iteration.

5.3 Experiment Setup

Alpha Data ADM-PCIE-7V3 board with a Xilinx Virtex-7 FPGA and 16GB device memory is used as the platform to validate our optimization framework. The FPGA board is connected to a host via PCI-e 3.0 X8 interface. Xilinx SDAccel 2016.2 is used as the OpenCL toolchain to synthesize OpenCL kernels onto FPGA. The operating frequency of all the benchmarks is set to be 200MHz, and all the benchmarks used in this work could be successfully synthesized under this frequency.

The evaluated OpenCL-based stencil benchmarks are from Polybench [22], Rodinia [23, 24], and Parboil [25] benchmark suites as shown in Table 2. The benchmarks are distinct in terms of stencil structures (e.g., dimension, size, etc) and the ratio of computation to global memory access intensity. For each benchmark, the execution time on FPGA is measured using the dynamic profiling tools provided by SDAccel.

Table 2: Stencil Benchmark Suite Description.

Benchmark	Source	Input Size	#Iterations
Jacobi-1D	Polybench [22]	131072	1024
Jacobi-2D	Polybench [22]	2048 × 2048	1024
Jacobi-3D	Parboil [25]	1024 × 1024 × 1024	1024
HotSpot-2D	Rodinia [23, 24]	4096 × 4096	1000
HotSpot-3D	Rodinia [23, 24]	4096 × 4096 × 128	1000
FDTD-2D	Polybench [22]	2048 × 2048	500
FDTD-3D	Polybench [22]	2048 × 2048 × 2048	500

5.4 Performance Results

Table 3 presents the performance speedup for all the benchmarks. For the baseline, we use the best design by exploring the design space of iteration fusion depth, tile size, and the number of simultaneous executing tiles (parallelism) [10]. In order to demonstrate the resource efficiency of our designs, we constrain them by the hardware size of the baseline [10]. Therefore, for our designs, we only vary the iteration fusion depth and tile size[1] but keep the parallelism same to the baseline. For our design, the optimal iteration depth and workload balancing factors are identified using our performance model. Overall, the performance speedup for the proposed Heterogeneous design ranges from 1.19X to 2.05X (on average 1.65X). Moreover, we can find that for each type of benchmark (Jacobi, HotSpot, or FDTD), the higher dimension the stencil has, the higher performance speedup can be achieved by our techniques. This is because the redundant computation increases exponentially with the number of dimensions, which gives larger optimization space for our techniques.

The performance speedup achieved by our designs lies in three folds, (1) eliminated redundant computation and global memory transfers, (2) increased iteration fusion depth, and (3) more balanced workload distribution. Figure 6 shows the execution time breakdown of different designs using *Jacobi-2D* and *Jacobi-3D* as case studies. For *Jacobi-2D*, our designs completely eliminate the redundant computation and memory transfer time, which are 17% and 6% of the overall execution time in the baseline design, respectively. For *Jacobi-3D*, our designs save more as the redundancy of the baseline increases with the dimension of the stencils. The saved memory storage also enables us to increase the depth of iteration fusion. As shown in Table 3, the optimal iteration fusion depth increases for all the benchmarks. The increased fused iteration number also helps to reduce the global memory transfer time. The workload balancing technique helps to balance the workload among the tiles and thus reduce the waiting time due to synchronization barrier. On average, the workload balancing techniques helps to reduce 9% waiting time.

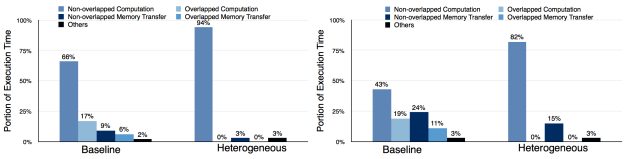
5.5 Resource Utilization Analysis

Table 3 compares the total resource (FF, LUT, DSP, and BRAM) utilization of different techniques. Since we use a uniform parallelism for the designs, the DSPs consumed are the same. The BRAM utilization is reduced by 8%-25% for the proposed heterogeneous design. The main contribution of this reduction comes from the pipe-based data sharing. Although the extra pipes also consume BRAMs, they still cannot offset the large benefit brought by data sharing. The number of FFs and LUTs are reduced by 8%-24% for heterogeneous designs. The reduction of FF and LUT

[1]The tile size shown in Table 3 of the heterogeneous design is the tile size of the slowest kernel.

Table 3: Experimental Results of Stencil Benchmark Suite.

Benchmark	Optimization	Optimization Parameters			Total Resource Utilization				Perf.
		#Fused Iter.	Tile Size	Parallelism	FF	LUT	DSP	BRAM	
Jacobi-1D	Baseline	128	4096	16	54864	79920	80	544	1
	Heterogeneous	512	4096	16	43896	62580	80	396	1.19
Jacobi-2D	Baseline	32	128×128	4×4	240016	343184	1792	1170	1
	Heterogeneous	63	120×120	4×4	191276	287955	1792	996	1.58
Jacobi-3D	Baseline	6	$16 \times 32 \times 32$	$4 \times 2 \times 2$	264026	367217	1802	1170	1
	Heterogeneous	16	$16 \times 28 \times 28$	$4 \times 2 \times 2$	237846	335951	1802	796	2.05
HotSpot-2D	Baseline	32	256×256	4×4	259040	251936	1920	1320	1
	Heterogeneous	69	248×248	4×4	233375	217197	1920	1081	1.35
HotSpot-3D	Baseline	6	$32 \times 32 \times 32$	$4 \times 2 \times 2$	225259	236664	1747	1260	1
	Heterogeneous	16	$30 \times 30 \times 30$	$4 \times 2 \times 2$	199625	207853	1747	1162	1.97
FDTD-2D	Baseline	12	64×64	4×4	104247	149457	324	560	1
	Heterogeneous	23	60×60	4×4	86872	131102	324	427	1.48
FDTD-3D	Baseline	4	$16 \times 32 \times 16$	$2 \times 4 \times 2$	149078	203266	518	952	1
	Heterogeneous	10	$14 \times 32 \times 15$	$2 \times 4 \times 2$	137632	176874	518	835	1.90



(a) Jacobi-2D (b) Jacobi-3D

Figure 6: Execution time breakdown.

utilization is directly related to the decreased BRAM consumption because large OpenCL data array implemented on FPGA needs multiplexers and registers to bundle BRAMs which requires LUTs and FFs.

5.6 Model Validation

In section 4, we propose a performance model for estimating execution time of iterative stencil algorithms in terms of clock cycles. Here, we verify our performance model by varying the number of the fused iterations as shown in Figure 7.

Overall, the model gives a highly accurate prediction. On average, the prediction error is 12%. For each benchmark, the optimal number of fused iterations found by our performance model is exactly the same as actual optimal value. But we notice that performance model underestimates the actual execution time. The underestimation is mainly due to the underestimated kernel launch time. In a real implementation of multiple kernels on FPGA, although multiple kernels execute in parallel, there exist a delay for the kernel launch. In other words, the kernels will be launched sequentially with a delay between adjacent kernel launches as shown in Figure 4. We do not model this kernel launch delay, which leads to the underestimated results. Nevertheless, the performance model captures the overall performance scaling trend for all the benchmarks.

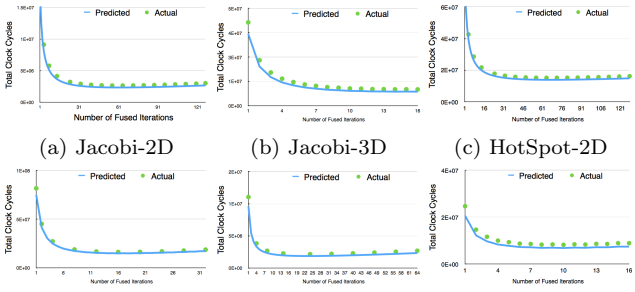
6. CONCLUSION

In this paper, we propose an efficient framework that automatically synthesizes highly optimized iterative stencil algorithms on FPGAs. An analytical performance model is built to perform fast design space exploration at the high level. A commercial OpenCL-to-FPGA toolchain is leveraged to generate accelerator design for FPGA. We first propose an efficient data sharing mechanism by utilizing OpenCL pipes to improve computation and global memory access efficiency. We then propose a workload balancing technique to balance the computation among different tiles. Experiments using a wide range of iterative stencil applications demonstrate that our heterogeneous implementations achieve 1.65X performance speedup on average but with less hardware resource compared to the state-of-the-art.

7. ACKNOWLEDGMENTS

This work was supported by the Natural Science Foundation of China (No. 61672048). We thank all the anonymous reviewers for their feedback.

8. REFERENCES



(a) Jacobi-2D (b) Jacobi-3D (c) HotSpot-2D

(d) HotSpot-3D (e) FDTD-2D (f) FDTD-3D

Figure 7: Validation of Performance Model.

[1] W. Huang et al., "Compact Thermal Modeling for Temperature-aware Design," in *DAC'04*.

[2] A. Akin et al., "A high-performance parallel implementation of the Chambolle algorithm," in *DATE'11*.

[3] G. L. G. Sleijpen et al., "A Jacobi–Davidson Iteration Method for Linear Eigenvalue Problems," *SIAM*, vol. 42, June 2000.

[4] A. Canis et al., "LegUp: High-level Synthesis for FPGA-based Processor/Accelerator Systems," in *FPGA'11*.

[5] J. Cong et al., "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *TCAD*, vol. 30, no. 4, pp. 473–491, 2011.

[6] J. Cong et al., "An Efficient and Versatile Scheduling Algorithm Based on SDC Formulation," in *DAC'06*.

[7] Y. Liang et al., "High-level Synthesis: Productivity, Performance, and Software Constraints," *JECE*, 2012.

[8] A. Putnam et al., "A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services," in *ISCA'14*.

[9] J. Ouyang et al., "SDA: Software-Defined Accelerator for Large-Scale DNN Systems," in *HotChips'26*.

[10] A. A. Nacci et al., "A High-level Synthesis Flow for the Implementation of Iterative Stencil Loop Algorithms on FPGA Devices," in *DAC'13*.

[11] L. Renganarayana et al., "Positivity, Posynomials and Tile Size Selection," in *SC'08*.

[12] J. Meng et al., "Performance Modeling and Automatic Ghost Zone Optimization for Iterative Stencil Loops on GPUs," in *ICS'09*.

[13] M. Christen et al., "PATUS: A Code Generation and Autotuning Framework for Parallel Iterative Stencil Computations on Modern Microarchitectures," in *IPDPS'11*.

[14] L. Renganarayana et al., "Towards Optimal Multi-level Tiling for Stencil Computations," in *IPDPS'07*.

[15] S. Krishnamoorthy et al., "Effective Automatic Parallelization of Stencil Computations," in *PLDI'07*.

[16] J. Fowers et al., "A Performance and Energy Comparison of FPGAs, GPUs, and Multicores for Sliding-window Applications," in *FPGA '12*.

[17] D. I. Moldovan and J. A. B. Fortes, "Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays," *IEEE Transactions on Computers*, 1986.

[18] J. Cong et al., "An Optimal Microarchitecture for Stencil Computation Acceleration Based on Non-Uniform Partitioning of Data Reuse Buffers," in *DAC'14*.

[19] Q. Xiao et al., "Exploring Heterogeneous Algorithms for Accelerating Deep Convolutional Neural Networks on FPGAs," in *DAC'17*.

[20] I. Beretta et al., "Parallelizing the Chambolle Algorithm for Performance-Optimized Mapping on FPGA Devices," *TECS*, vol. 15, pp. 44:1–44:27, Mar. 2016.

[21] S. Wang et al., "FlexCL: An Analytical Performance Model for OpenCL Workloads on Flexible FPGAs," in *DAC'17*.

[22] S. Grauer-Gray et al., "Auto-tuning a high-level language targeted to GPU codes," in *InPar'12*.

[23] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," in *IISWC'09*.

[24] S. Che et al., "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," in *IISWC'10*, pp. 1–11, 2010.

[25] J. A. Stratton et al., "Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing," Tech. Rep. IMPACT-12-01, UIUC, 2012.